
Certification of programs with computational effects

Burak Ekici

May 12, 2014

Abstract. In purely functional programming languages imperative features, more generally computational effects are prohibited. However, non-functional languages do involve effects. The theory of decorated logic provides a rigorous formalism (with a refinement in operation signatures) for proving program properties with respect to computational effects. The aim of this thesis is to first develop Coq libraries and tools for verifying program properties in decorated settings associated with several effects: states, local state, exceptions, non-termination, etc. Then, these tools will be combined to deal with several effects.

The syntax of any programming language defines the set of rules that a programmer may utilize during coding progress while its semantics stand for proving program properties. In a purely functional programming language, a term f with an argument of type X and a result of type Y (which may be written $f : X \rightarrow Y$) is denotationally interpreted as a function f between the sets $[X]$ and $[Y]$. It follows that, when an operation has several arguments, they can be evaluated in parallel, or in any order. It is possible to interpret a purely functional programming language via categorical semantics based on *cartesian closed categories*. The word “*cartesian*” here refers to the categorical products, which are interpreted as cartesian products of sets and used for dealing with pairs of arguments. The logical semantics of such a language defines a set of rules that may be used for proving properties of programs. But non-functional programming languages such as IMP, C or Java do include computational effects. For instance, a C or IMP function may modify the state structure and a Java function may throw an exception during the computation. Such operations are examples of computational effects. To cope with them, [Dumas:2011a] provides a canonical approach. It indeed uses **decorations** on the main operators of the effect (*with superscripts between parentheses*) instead of mentioning the effect itself. I.e., any *state accessor function* f , seen as $X \times S \rightarrow Y$ can be interpreted as $f^{(1)} : X \rightarrow Y$ where X and Y are sets with the distinguished set of states S and with *cartesian product operator*, “ \times ”. Thus, this approach proposes a refinement in the operator signatures by keeping them closer to syntax where there is no effect appearance but instead **decorations**. Technical details of the refinement are based on *category theoretical objects*: see procedure 0 .

Accordingly, my PhD study simply focuses on formal models of computational effects (with effect combinations) through *decorated logic* over *cartesian effect categories* [Dumas:2011a].

Besides, we develop Coq libraries to verify program properties with respect to effects in question.

- 1: **procedure** DEC. ACC. (\mathbb{C}, \mathbb{S}) Cartesian effect category \mathbb{C} , distinguished states object \mathbb{S}
- 2: $\Phi: \mathbb{C} \rightarrow \mathbb{C}$ is an endo-functor defined as follows:

$$\Phi(X) = X \times \mathbb{S} \quad \text{and} \quad \Phi(f: X \rightarrow Y) = (f \times \text{id}_{\mathbb{S}}): X \times \mathbb{S} \rightarrow Y \times \mathbb{S}$$
- 3: $\text{cM}(\Phi, \delta: \Phi \Rightarrow \Phi^2, \epsilon: \Phi \Rightarrow \text{id}_{\mathbb{C}})$ is the **states comonad** with following settings:

$$\delta_X: X \times \mathbb{S} \rightarrow X \times \mathbb{S} \times \mathbb{S} \quad \text{and} \quad \epsilon_X: X \times \mathbb{S} \rightarrow X$$

$$(x, s) \mapsto (x, s, s) \quad \text{and} \quad (x, s) \mapsto x$$
- 4: \mathbb{C}_1 is the **coKleisli** category of cM over \mathbb{C} defined as:

$$\text{Obj}(\mathbb{C}_1) = \text{Obj}(\mathbb{C}), \quad \text{Hom}_{(\mathbb{C})}(X, Y) = \text{Hom}_{(\mathbb{C}_1)}(\Phi X, Y) \quad g^{(1)} \circ_{\mathbb{C}_1} f^{(1)} = g_0 \circ_{\mathbb{C}} \Phi(f_0) \circ_{\mathbb{C}} \delta$$
- 5: **return** Any accessor $f_0: X \times \mathbb{S} \rightarrow Y \in \text{Hom}_{(\mathbb{C})}$ is interpreted as $f^{(1)}: X \rightarrow Y \in \text{Hom}_{(\mathbb{C}_1)}$
- 6: **end procedure**

Following is the brief description to the procedure 0 : the *endofunctor*, Φ on *cartesian effect category of sets*, \mathbb{C} , is a *comonad* with *natural transformations* δ and ϵ . Thus, this triple yields in the existence of the *coKleisli category*, \mathbb{C}_1 . Therefore, any *impure* function $f^{(1)}: X \rightarrow Y \in \mathbb{C}_1$ is the interpretation of $f_0: X \times \mathbb{S} \rightarrow Y \in \mathbb{C}$ representing *accessors*.

To interpret *modifiers*, the *endofunctor*, Φ_1 (having the same settings with Φ) on category, \mathbb{C}_1 with compatible *natural transformations* μ and η is proven as a *monad* on which a *Kleisli category*, \mathbb{C}_2 is built. Similarly, any *impure* function $f^{(2)}: X \rightarrow Y \in \mathbb{C}_2$ is the interpretation of $f_0: X \times \mathbb{S} \rightarrow Y \times \mathbb{S} \in \mathbb{C}$ thus representing *modifiers*. Additional to those syntactical tricks, we also have an *algebra* for *states effect* mainly based on *monadic equational logic*, *categorical products* and some *observational properties*. The *hierarchy rules* define the transition between different sorts of operators: a *pure function* can be seen as an *accessor* and similarly an *accessor* function can be seen as a *modifier*. For details, see [Dumas:2012:states].

A generic *Coq library* formalizing the *states effect* in above given settings was developed and detailed in [Dumas:2014:coqstates]. The main idea here is to prove the 7 primitive properties of the states structure proposed by [Plotkin&Power:2002]. This provides an environment in which programmers are enabled to prove program properties through already proven lemmas with respect to *states effect*. Those proofs become crucial when the order of evaluation is not specified or more generally when parallelization comes into play [Lucassen&Gifford:1988]. To check the *soundness* of this proof system, we have specialized the generic library for the case of IMP language and proven equalities between some IMP programs involving *terminating loops* and *conditionals*. See the link .

Thanks to the *duality* between *exceptions* and *states* [Dumas:2012:duality], all the syntactical tricks together with the algebra for states have been dualized for exceptions. I.e., *lookup* operation on the state is dual to *tagging* an exception. Thus, an environment to cope with *exceptions effect* has been ensued and developed in Coq. Here is the link to the generic library for *exceptions*. In addition, we have combined mentioned effects through the following non-canonical way: just composing *functors* and merging the *algebras*. In order to see its *soundness*, we have specialized the library for $\{\text{IMP}+\text{Exc}\}^1$ language. It can be found through the link .

¹The fact that IMP has no *exception handling mechanism*, we simply defined *throw* operation and *try-catch* block as additional commands. The resulting language is called $\{\text{IMP}+\text{Exc}\}$. It is developed in Coq syntax but in IMP semantics where all commands have $\mathbb{1} \rightarrow \mathbb{1}$ type.

It apparently follows that in this library *equational proofs* between $\{\text{IMP+Exc}\}$ programs can be stated including both *states* and *exceptions effects*.

Considering future directions, we are planning to introduce a *canonical framework* first to combine *states* and *exceptions* in *decorated settings*. Then, we will make attempts to generalize the idea to the other ones. In the mean time, the comparison with *monad transformers* is planned to be stated.

REFERENCES

- [Dumas:2014:coqstates] Jean-Guillaume Dumas and Dominique Duval and Burak Ekici and Damien Pous. Formal verification in Coq of program properties involving the global state effect. In JFLA, Fréjus, 2014.
- [Dumas:2012:states] Jean-Guillaume Dumas, Dominique Duval, Laurent Fousse, and Jean-Claude Reynaud. Decorated proofs for computational effects: States. In *ACCAT*, pages 45–59, 2012.
- [Dumas:2011a] Jean-Guillaume Dumas, Dominique Duval, Jean-Claude Reynaud. Cartesian effect categories are Freyd-categories. *Journal of Symbolic Computation* 46, p. 272-293 (2011).
- [Dumas:2012:duality] Jean-Guillaume Dumas, Dominique Duval, Laurent Fousse and Jean-Claude Reynaud. *Journal of Mathematical Structures in Computer Science* 22, p. 719-722(2012).
- [Plotkin&Power:2002] Gordon D. Plotkin, John Power. Notions of Computation Determine Monads. FoSSaCS 2002. Springer-Verlag Lecture Notes in Computer Science 2303, p. 342-356 (2002).
- [Lucassen&Gifford:1988] J. M. Lucassen and D. K. Gifford. Polymorphic effect systems. In J. Ferrante and P. Mager, editors, *POPL*, pages 47–57. ACM, 1988. ISBN 0-89791-252-7.